# Verifying Hybrid Systems Modeled as Timed Automata: A Case Study*

**Presented at HART '97, Grenoble, France, March 26-28, 1997**

Myla Archer and Constance Heitmeyer

Code 5546, Naval Research Laboratory, Washington, DC 20375
{archer, heitmeyer}@itd.nrl.navy.mil

**Abstract.** Verifying properties of hybrid systems can be highly complex. To reduce the effort required to produce a correct proof, the use of mechanical verification techniques is promising. Recently, we extended a mechanical verification system, originally developed to reason about deterministic real-time automata, to verify properties of hybrid systems. To evaluate our approach, we applied our extended proof system to a solution, based on the Lynch-Vaandrager timed automata model, of the Steam Boiler Controller problem, a hybrid systems benchmark. This paper reviews our mechanical verification system, which builds on SRI's Prototype Verification System (PVS), and describes the features we added to handle hybrid systems. It also discusses some errors we detected in applying our system to the benchmark problem. We conclude with a summary of insights we acquired in using our system to specify and verify hybrid systems.

## 1 Introduction

Researchers have proposed many innovative formal methods for developing real-time systems [9]. Such methods can give system developers and customers greater confidence that real-time systems satisfy their requirements, especially their critical requirements. However, applying formal methods to practical systems requires the solution of several challenging problems, e.g., how to make formal descriptions and formal proofs understandable to developers and how to design software tools in support of formal methods that are usable by developers.

We are building a mechanized system for specifying and reasoning about real-time systems that is designed to address these challenging problems. Our approach is to build formal reasoning tools that are customized for specifying and verifying systems represented in terms of a specific mathematical model. In [2], we describe how we are using the mechanical proof system PVS [18, 19] to support formal specification and verification of systems modeled as Lynch-Vaandrager timed automata [15, 14]. Reference [2] also presents the results of a case study in which we applied our method to prove invariant properties of a solution to the Generalized Railroad Crossing (GRC) problem [7].

Our system provides mechanical assistance that allows humans to specify and reason about real-time systems in a direct manner. To specify a particular timed automaton, the user fills in a template provided by our system. Then, he or she uses our system, a version of PVS augmented with a set of specialized PVS proof strategies, to verify properties of the automaton. Use of our system for

* This work is funded by the Office of Naval Research. URLs for the authors are http://www.itd.nrl.navy.mil/ITD/5540/personnel/{archer, heitmeyer}.html

# Report Documentation Page

*Form Approved*
*OMB No. 0704-0188*



| 1. REPORT DATE | 2. REPORT TYPE | 3. DATES COVERED |
|---|---|---|
| **1997** | | **00-00-1997 to 00-00-1997** |

| 4. TITLE AND SUBTITLE | 5a. CONTRACT NUMBER |
|---|---|
| **Verifying Hybrid Systems Modeled as Timed Automata: A Case Study** | 5b. GRANT NUMBER |
| | 5c. PROGRAM ELEMENT NUMBER |
| 6. AUTHOR(S) | 5d. PROJECT NUMBER |
| | 5e. TASK NUMBER |
| | 5f. WORK UNIT NUMBER |
| 7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Naval Research Laboratory,Code 5546,4555 Overlook Avenue, SW,Washington,DC,20375** | 8. PERFORMING ORGANIZATION REPORT NUMBER |
| 9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) | 10. SPONSOR/MONITOR'S ACRONYM(S) |
| | 11. SPONSOR/MONITOR'S REPORT NUMBER(S) |

12. DISTRIBUTION/AVAILABILITY STATEMENT
**Approved for public release; distribution unlimited**

13. SUPPLEMENTARY NOTES

14. ABSTRACT

15. SUBJECT TERMS

| 16. SECURITY CLASSIFICATION OF: | | | 17. LIMITATION OF ABSTRACT | 18. NUMBER OF PAGES | 19a. NAME OF RESPONSIBLE PERSON |
|---|---|---|---|---|---|
| a. REPORT **unclassified** | b. ABSTRACT **unclassified** | c. THIS PAGE **unclassified** | | **15** | |

specification and verification is usually quite straightforward because our system provides both a structure and a set of specialized theories useful in constructing timed automata models and proving properties about them. By focusing on a particular mathematical model—the timed automata model—our system allows a user to reason within a specialized mathematical framework. The user need not master the base logic and the complete user interface of the underlying proof system, PVS.

The results of our initial study proved encouraging. All of the hand proofs of invariant properties in the GRC solution were translated into corresponding PVS proofs with a very similar structure. Moreover, most of the PVS proofs could be done using our specialized strategies alone. Neither the time required to enter the specifications using our template nor the time required to check the proofs of state invariants was excessive.

While our initial study only involved deterministic automata, we recently demonstrated that the same approach could be applied to hybrid automata, where the effects of time passage and other events can be nondeterministic. To study the utility of our proof techniques for verifying hybrid systems, we applied them to a solution of the Steam Boiler Controller problem described in [12]. In verifying this application, we investigated the following issues:

- How should nondeterminism be modeled?
- Is it practical to use an automatic theorem prover to reason about nonlinear real arithmetic?
- Can the timed automata template and proof strategies developed in our earlier case study be extended to handle the more general problem of hybrid automata?
- How suitable is PVS for verifying properties of hybrid systems?

In verifying the solution in [12], we detected several errors. While most of these errors are minor and easily fixed, at least two of the errors are errors in reasoning, and their correction is nontrivial.

Like other approaches to verifying real-time systems, such as SMV [16, 5], HyTech [10], and COSPAN [11], our approach is based on a formal automata model. Moreover, like these other approaches, our methods can be used to prove properties of particular automata and, like COSPAN, to prove simulations between automata.[1] However, our approach is different from other approaches in several ways. First, the properties we prove are expressed in a standard logic with universal and existential quantification. This is in contrast to most other approaches, where the properties to be proved are expressed either in a temporal logic, such as CTL or Ictl, or in terms of automata. By using standard notations and standard logics and by providing templates for developing specifications, we largely eliminate the need for the special notations, logics, etc., required by other verification systems. Second, unlike other automata-based methods, the generation of proofs in our method is not completely automatic. Rather, our method uses a mechanical proof system to check the validity of hand proofs that use deductive reasoning.

[1] We have designed support for verifying simulation proofs, but implementation of this support requires some improvements to PVS.

Interaction with an automatic proof system does demand a higher level of sophistication from the user. But by supporting reasoning about automata at a high level of abstraction, we can prove more powerful results than is possible with tools requiring more concrete descriptions of automata. Moreover, our approach avoids the state explosion problem inherent in other automata-based approaches and also provides considerable feedback when an error is detected. Such feedback is extremely useful in correcting errors.

Section 2 reviews the timed automata model, PVS, and the template and tools we developed in our earlier verification of the GRC. Section 3 introduces the Steam Boiler Controller problem, presents the main hybrid automaton specified in [12], and discusses the techniques we used to adapt our template to proving properties of hybrid automata. Section 4 presents our major results—the errors we detected in applying our system to the Boiler Controller problem and the time and effort we needed to adapt and apply our methods. Section 5 presents an example of a hand proof and the corresponding PVS proof for a state invariant whose proof involves both nondeterminism and nonlinear real arithmetic. Section 6 presents some conclusions about the usefulness of our methods for verifying hybrid automata.

## 2 Background

### 2.1 The Timed Automata Model

The formal model used in the specification of the Boiler Controller problem and its solution [12] represents both the computer system controller and its environment as Lynch-Vaandrager *timed automata*. A timed automaton is a very general automaton, i.e., a labeled transition system. It need not be finite-state: for example, the state can contain real-valued information, such as the current time, the boiler's water level, and the boiler's steam rate. This makes timed automata suitable for modeling not only computer systems but also real-world quantities, such as water levels and steam rates. The timed automata model describes a system as a set of timed automata, interacting by means of common actions. In the Boiler Controller solution presented in [12], separate timed automata represent the boiler and the controller; the common actions are sensors reporting the water level, steam rate, and number of active pumps in the boiler, and actuators controlling whether to stop the boiler and when the next sensor action is scheduled. The definition of timed automata below, which is based on the definitions in [8, 7], was used in our earlier case study [2]. It is a special case of Lynch-Vaandrager timed automata, which requires the next-state relation, $steps(A)$, to be a function. How to modify the definition to handle the nondeterminism inherent in hybrid automata was one of the issues addressed in our current study.

A *timed automaton* $A$ consists of five components:

- $states(A)$, a (finite or infinite) set of states.
- $start(A) \subseteq states(A)$, a nonempty (finite or infinite) set of start states.
- A mapping $now$ from $states(A)$ to $R^{\geq 0}$, the non-negative real numbers.
- $acts(A)$, a set of actions (or events), which include special *time-passage* actions $\nu(\Delta t)$, where $\Delta t$ is a positive real number, and *non-time-passage* actions, classified as *input* and *output* actions.
- $steps(A) : states(A) \times acts(A) \rightarrow states(A)$, a partial function that defines the possible steps (i.e., transitions).

### 2.2 PVS

PVS (Prototype Verification System) [19] is a specification and verification environment developed by SRI International's Computer Science Laboratory. In contrast to other widely used proof systems, such as HOL [6] and the Boyer-Moore theorem prover [4], PVS supports *both* a highly expressive specification language and an interactive theorem prover in which most low-level proof steps are automated. The system provides a specification language, a parser, a type checker, and an interactive proof checker. The PVS specification language is based on a richly typed higher-order logic that permits a type checker to catch a number of semantic errors in specifications. The PVS prover provides a set of inference steps that can be used to reduce a proof goal to simpler subgoals that can be discharged automatically by the primitive proof steps of the prover. The primitive proof steps incorporate arithmetic and equality decision procedures, automatic rewriting, and BDD-based boolean simplification.

In addition to primitive proof steps, PVS supports more complex proof steps called *strategies*, which can be invoked just like any other proof step in PVS. Strategies may be defined using primitive proof steps, applicative Lisp code, and other strategies, and may be built-in or user-defined.

### 2.3 A Template For Specifying Timed Automata in PVS

Our template for specifying Lynch-Vaandrager timed automata provides a standard organization for an automaton. To define a timed automaton, the user supplies the following six components:

- declarations of the non-time actions,
- a type for the "basic state" (usually a record type) representing the state variables,
- any arbitrary state predicate that restricts the set of states (the default is **true**),
- the preconditions for all transitions,
- the effects of all transitions, and
- the set of start states.

In addition, the user may optionally supply

- declarations of important constants,
- an axiom listing any relations assumed among the constants, and
- any additional declarations or axioms desired.

To support mechanical reasoning about timed automata using proof steps that mimic human proof steps, we also provide an appropriate set of PVS strategies, based on a set of standard theories and certain template conventions. For example, the induction strategy, which is used to prove state invariants, is based on a standard automaton theory called **machine**. To reason about the arithmetic of time, we have developed a special theory called **time_thy** and an associated simplification strategy called **TIME_ETC_SIMP** for time values that can be either non-negative real values or $\infty$.

## 3 Specifying and Reasoning About Hybrid Automata

Like others, we use the term *hybrid automaton* to describe a state machine which has both continuous and discrete components. Since hybrid automata are used to model physical systems controlled by a discrete computer system, the laws of physics affect their behavior. Because changes in measurable properties of the physical systems depend on environmental factors, automata models of these systems typically have some nondeterministic transitions. Thus, hybrid

automata differ from deterministic, discrete automata in two major ways: their behavior is nondeterministic and reasoning about their transitions often leads to complex computations involving nonlinear real arithmetic.

Strictly speaking, the automata in our case study of the GRC problem [2] are hybrid automata by the above definition. However, their description involved neither nondeterminism nor nonlinear arithmetic. For the example in the current case study, it was necessary to extend our previous methods to handle both of these features.

### 3.1 Reasoning About Hybrid Automata

**Handling Nondeterminism.** To specify and to reason about nondeterministic automata using an automated theorem prover, one needs to describe the nondeterministic transitions. An obvious approach is to represent the transitions as a relation. In an initial experiment, we encoded the transitions in a deterministic automaton from our initial study [2] as a relation and redid the proofs of state invariants up to and including the Safety Property, the major theorem. Because the selected timed automaton is deterministic, the transition relation was expressed as an equality between the result state and a case expression involving the action and the initial state. The new versions of the state invariant proofs were quite similar to the corresponding proofs in [2] , except for two important differences. First, they took about twice as long to execute. Second, frequent human intervention was required to substitute the value of the result state where it was needed. Human intervention was required even in the "trivial" branches of induction proofs that our system previously handled automatically. Clearly, this situation would only worsen in the case of true nondeterminism.

In the case of hybrid automata, not all transitions are nondeterministic, and among those that are, not all parts of the state change nondeterministically. Our ultimate solution, which took advantage of this fact, uses PVS's implementation "epsilon" of Hilbert's choice operator $\epsilon$ in expressing the transition relation as a function. The choice operator $\epsilon$ is defined as follows. Let $P : \tau \rightarrow \mathbf{bool}$ be any predicate on a nonempty type $\tau$. Then $\epsilon(P)$ has two known properties. First, $\epsilon(P)$ is an element of type $\tau$. Second, the $\epsilon$*–axiom* holds, namely: if there is an element of type $\tau$ that satisfies $P$, then $P$ holds for $\epsilon(P)$, i.e., $P(\epsilon(P)) = \mathbf{true}$.

Two features of $\epsilon(P)$ should be noted. First, it is always defined, whether or not $P$ can be satisfied, and second, it is deterministic. These characteristics need to be remembered when one is using $\epsilon$ to reason about nondeterministic automata. Reasoning about such automata using $\epsilon$ is sound, provided 1) one does not try to draw conclusions about existence of an action's result state *satisfying the specified constraints* from the fact that the action is defined, and 2) one does not draw conclusions about the equality of states reached by identical action sequences. If such inferences are avoided, any specification or proof errors uncovered using $\epsilon$ will be genuine errors.

A careful use of $\epsilon$ that avoids these two dangers will simulate what we consider to be a better solution: implement "ANY" and "SOME" quantifiers in PVS that can be skolemized and instantiated in fashions analogous to "FORALL" and "EXISTS" in PVS. An "ANY" quantifier would avoid at least problem 2), since constants arising from its multiple skolemization could never be proved equal. A solution to problem 1) is more difficult in PVS, since PVS does not allow

```
real_thy: THEORY
  BEGIN
    nonnegreal: TYPE = {r:real | 0 <= r};
  % posreal_mult_closed: LEMMA (FORALL (x,y:real): (x > 0 & y > 0) => x*y > 0);
    nonnegreal_mult_closed: LEMMA (FORALL (x,y:real): (x >= 0 & y >= 0) => x*y >= 0);
    greater_eq_nonnegmult_closed: LEMMA (FORALL (x,y,z:real): (x >= 0 & y >= z) => x*y >= x*z);
  END real_thy
```

**Fig. 1.** A (Partial) Theory of Real Numbers

partial functions; "ANY" would have to behave more or less like $\epsilon$ with respect to definedness.

**Handling Nonlinear Real Arithmetic.** Computation in several of our PVS proofs involved nonlinear real expressions. Existing PVS decision procedures are able to handle some simple cases of such reasoning by expanding terms into a normal form and matching identical terms. When the reasoning is more complex, PVS needs help. Though it does not always happen, the simplifications done by the PVS decision procedures can sometimes confuse the argument in the proof. There are two approaches available to solve these problems. First, we can provide PVS with a list of facts about real numbers. Second, we can name subterms in a manner that prevents PVS from overdoing simplification.

Providing support for convenient use of the second approach must await planned enhancements to PVS. However, implementing the first approach is more straightforward. We have added a line to our template that imports the theory **real_thy**, which contains useful standard definitions and lemmas about real numbers. Figure 1 shows the subset of that theory that was helpful in completing the real arithmetic reasoning in the proofs in the current case study. We note that, unlike [20], our goal is *not* to encode and use deep facts about real analysis (such as the properties of integrals), since 1) these properties are well understood, and 2) it is usually possible to isolate any application of them (and the proof that it was done correctly) from the rest of the specification correctness proof. Instead, we aim to compile a list of facts that can eventually be used automatically in an improved "real-arithmetic" PVS strategy.

### 3.2 The Steam Boiler Controller Problem

The Steam Boiler Controller problem, which is intended to provide a realistic benchmark for comparing different real-time formalisms, was defined by J.-R. Abrial et al. in [1] and is derived from a competition problem previously posed by Lt-Col. J.C. Bauer for the Institute for Risk Research at the University of Waterloo in Canada. Below is the condensed and informal description of the problem that appears in [12]:

> The physical plant consists of a steam boiler. Conceptually, this boiler is heated (e.g., by nuclear fuel) and the water in the boiler evaporates into steam and escapes the boiler to drive, e.g., a generator (this part is of no concern to the problem). The amount of heat and, therefore, the amount of steam changes without any considered control. Nevertheless, the safety of the boiler depends on a bounded water level ($\mathbf{q}$) in the boiler and steam rate ($\mathbf{v}$) at its exit. A set of four equal pumps may supply water to compensate for the steam that leaves the boiler. These four pumps can be activated or stopped by the controller system. The controller reacts to the information of two sensors, the

water level sensor and the steam rate sensor, and both may fail. Moreover, the controller can deduce from a pump monitor whether the pumps are working correctly. Sensor data are transferred to the controller system periodically. The controller reacts instantaneously with a new setting for the pumps (**pr_new**) or decides to shut down the boiler system (**stop**).

There are two basic time constants: First, the time between two consecutive sensor readings (denoted **I**) and, second, the delay time (**S**) until the reaction of the controller causes consequences in the boiler. The latter delay time usually represents a worst case accumulation of sensor reading delay, calculation time in the controller, message delivery time, reaction time of the pumps, and other minor factors.

More precisely, the water level has two safety limits, one upper (denoted $\mathbf{M}_2$) and one lower limit (denoted $\mathbf{M}_1$). If the water level reaches either limit, there is just time enough to shut down the system before the probability of a catastrophe gets unacceptably high. The steam rate has an upper limit (denoted **W**) and, again, if this limit is reached the boiler must be stopped immediately. In addition the human operator has the possibility to activate the shutdown anytime.

Several automata are specified in [12]: a boiler, a simple controller, the combination of these into a combined system, and a combined system with a fault-tolerant controller. Figure 2 presents, essentially verbatim, the specification of the simple combined system given in [12].[2] In our study, we began by entering the details of this specification into our timed automaton template.

Figure 3 shows a fragment of our template's version of this specification. This fragment includes the definitions of two (parameterized) predicates **steam_rate_pred** and **water_level_pred** (shown bold face for readability) that capture the constraints on the new steam rate $v'$ and water level $q'$ in the effect of the time passage action $\nu(\Delta t)$, together with the $\nu(\Delta t)$ part of the definition of the template's transition function *trans* that encodes the effects of the actions defined in Figure 2. Appropriate applications of the $\epsilon$ operator in a LET construct allow the effect of the action $\nu(\Delta t)$ on state $s$ to be expressed both compactly and *carefully*.[3] All the nondeterministic state components in either the start state or in the transitions were similarly handled.

The major proof goal in [12] for the simple combined system was to establish two properties: either the steam rate has an acceptable value or the system is in the stop mode, and either the water level has an acceptable value or the system is in the stop mode. These two properties are expressed formally in [12] as:

**Theorem 1:** In all reachable states of the boiler system, $v < \mathbf{W}$ *or* $stop = \mathbf{true}$.

**Theorem 2:** In all reachable states of the boiler system, $\mathbf{M}_1 < q < \mathbf{M}_2$ *or* $stop = \mathbf{true}$.

Although neither property mentions time explicitly, the correctness of the specification with respect to these properties is established by proving a sequence of state invariants that depend on time and thus relies heavily upon timing restrictions.

[2] The meanings of several of the constants and variables are found in the above problem description. The Appendix defines the remaining constants and variables.

[3] The LET construct allows $\epsilon$ to be used in encoding the definition of any state in which two nondeterministic components are related (as is the case in the start state—where $q$ and $wl$, which satisfy the same predicate, must be equal—as well as in the result state of $\nu(\Delta t)$) without the determinism of the $\epsilon$ operator being taken for granted.

**State:**

*now*, a nonnegative real, initially 0
*do_sensor* : boolean, initially ***true***
*do_output, stopmode, stop* : boolean, initially ***false***
*pr, pr_new, pumps, px* : [0 .. ***#pumps***], initially 0
*error* : [0 .. *pr_new*], initially 0
*v, read* : nonnegative real, initially 0
*set* : nonnegative real, initially $\boldsymbol{S}$
*sr* : [0,$\boldsymbol{W}$], initially 0
*q, wl* : [0,$\boldsymbol{C}$], initially equal and $\gg \boldsymbol{M}_1$ and $\ll \boldsymbol{M}_2$

**Transitions:**

***actuator(e_stop,pset)***
**Precondition:**
$do\_output = \boldsymbol{true}$
$pset = px$
$e\_stop = stopmode$
**Effect:**
$do\_output' = \boldsymbol{false}$
$do\_sensor' = \boldsymbol{true}$
$pr\_new' = pset$
$stop' = e\_stop$
$read' = now + \boldsymbol{I}$

***controller***
**Precondition:**
***true***
**Effect:**
$0 \le px' \le \boldsymbol{\#pumps}$

$\nu(\Delta t)$
**Precondition:**
$stop = \boldsymbol{false}$
$now + \Delta t \le read$
$now + \Delta t \le set$
**Effect:**
$v - \boldsymbol{U}_2 * \Delta t \le v' \le v + \boldsymbol{U}_1 * \Delta t$
$q + pr * \boldsymbol{P} * \Delta t - \delta_{HIGH}(v, v', \Delta t) \le q'$
$q' \le q + pr * \boldsymbol{P} * \Delta t - \delta_{LOW}(v, v', \Delta t)$
$now' = now + \Delta t$

***sensor(s,w,p)***
**Precondition:**
$now = read$
$do\_sensor = \boldsymbol{true}$
$stop = \boldsymbol{false}$
$w = q$
$s = v$
$p = pr$
**Effect:**
$pumps' = p$
$do\_sensor' = \boldsymbol{false}$
$do\_output' = \boldsymbol{true}$
$sr' = s$
$wl' = w$

$if\ sr' \le \boldsymbol{W} - \boldsymbol{U}_1 * \boldsymbol{I}\ or$
$wl' \ge \boldsymbol{M}_2 - \boldsymbol{P} * (pumps' * \boldsymbol{S} + (max\_pumps\_after\_set) * (\boldsymbol{I} - \boldsymbol{S})) + min\_steam\_water(sr)\ or$
$wl' \le \boldsymbol{M}_1 - \boldsymbol{P} * (pumps' * \boldsymbol{S} + (min\_pumps\_after\_set) * (\boldsymbol{I} - \boldsymbol{S})) + max\_steam\_water(sr)$
$then\ stopmode' = \boldsymbol{true}$
$else\ stopmode' = \{\boldsymbol{true}, \boldsymbol{false}\}\ arbitrary$

***activate***
**Precondition:**
$now = set$
$stop = \boldsymbol{false}$
**Effect:**
$set' = read + \boldsymbol{S}$
$0 \le error' \le pr\_new$
$pr' = pr\_new - error$

**Fig. 2.** Initial Specification of the Boiler System

## 4 Results

**Extensions Required to Previous Methods.** Once the decision to model nondeterminism using the Hilbert $\epsilon$ was made, just two modifications to the system used in our earlier study [2] were required before the system could be applied to hybrid systems. First, some necessary lemmas about real arithmetic were identified and included in a new auxiliary theory **real_thy**. The other major modification involved $\epsilon$: we developed a technique for using $\epsilon$ in specifications (illustrated in Figure 3) and in proofs (illustrated in Figure 6).

**Errors Discovered.** Using our proof system to analyze the Boiler, Controller, and combined Boiler System specifications in [12] identified approximately a

```
steam_rate_pred(v_old:nonnegreal,delta_t:(fintime?))(v_new:nonnegreal):bool =
   v_old – U_2*dur(delta_t) <= v_new & v_new <= v_old + U_1*dur(delta_t);

water_level_pred(q_old:water_level,pr:num_pumps,v_old,v_new:nonnegreal,delta_t:(fintime?))
                 (q_new:water_level):bool =
   q_old + pr*P*dur(delta_t) – delta_HIGH(v_old,v_new,delta_t) <= q_new
 & q_new <= q_old + pr*P*dur(delta_t) – delta_LOW(v_old,v_new,delta_t);

trans (a:actions, s:states):states =
   CASES a OF
    nu(delta_t): LET new_v_part = epsilon(steam_rate_pred(v(s),delta_t)),
                     new_q_part = epsilon(water_level_pred(q(s),pr(s),v(s),new_v_part,delta_t))
                 IN s WITH [now := now(s) + delta_t,
                            basic := basic(s) WITH [v_part := new_v_part, q_part := new_q_part]],
    <... other action cases ...>
   ENDCASES;
```

**Fig. 3.** Specifying Nondeterminism Using $\epsilon$

dozen errors.[4] These errors were discovered during each of the three basic stages of the mechanization process: filling in the template, applying the PVS type checker, and checking the hand proofs of the invariants. The errors detected in filling in the template were usually due to vagueness in the original specification. For example, what is the meaning of the expressions "$q << \mathbf{M}_2$" and "$q >> \mathbf{M}_1$"? How is the water level variable $wl$ in the Controller assigned the value of the actual water level $q$ in the Boiler when the Controller is considered as a separate automaton?

In checking the hand proofs, several types of errors were found. The most common were missing assumptions about the constants. The second most common were simple typographical errors. All of these errors, as well as the errors due to vagueness, are easily fixed. To correct them, we determined what changes were needed to successfully complete the hand versions of the invariant proofs. While trivial, detecting and correcting these easily fixable errors does clearly lead to better specifications and better proofs.

Figure 4 shows an example of the type of feedback that occurs in the presence of an error. It shows the encoding of invariant lemma 3.3 of [12], and the goal of the sensor action proof branch generated by the induction strategy, with the precondition in hypothesis {-4} expanded. The goal is in the form of a PVS Gentzen-style *sequent*, where the conjunction of the hypotheses (numbered negatively) must be shown to imply the disjunction of the conclusions (numbered positively). Clearly, the proof on this branch would be valid if the inequality "sr_1 <= W – U_1 * I" in the condition in conclusion [2]—which is easily traceable to the definition of the sensor action in Figure 2—were reversed. A little thought shows that the first inequality in this definition is indeed backwards: one wants to prevent the steam rate $sr'$ from becoming too *large*, not too small.

In addition to the minor errors described above, two more serious errors—errors in reasoning—were discovered. How to correct these errors was not obvious. Like many of the minor errors, errors in reasoning were found when a dead

[4] A complete description of these errors can be found in [3].

```
Inv_3_3(s: states):bool =   (sr(s) + U_1*I < W)   OR   stopmode(s) = true;
lemma_3_3: LEMMA (FORALL (s:states): reachable(s) => Inv_3_3(s));
```

```
lemma_3_3.2 :
[−1]   reachable(s_1)
[−2]   (((sr_part(basic(s_1)) + U_1 * I) < W)   OR   stopmode_part(basic(s_1)))
[−3]   enabled_general(sensor(sr_1, w_1, p_1), s_1)
{−4}    now(s_1) = fintime(read(s_1))  AND  do_sensor(s_1)  AND  w_1 = q(s_1)  AND  sr_1 = v(s_1)
       AND  p_1 = pr(s_1) AND  NOT(stop(s_1))
|−−−−−−−
[1]   (U_1 * I + sr_1 < W)
[2]   IF (sr_1 <= W − U_1 * I
       OR ((w_1 >= ((((max(0, sr_part(basic(s_1))*I − U_2*I/2*I) + M_2) − (I*P*num_pumps)) − P*S*p_1)
                    + (P*S*num_pumps)))
       OR (w_1 <= (M_1 + sr_part(basic(s_1))*I + U_1*I/2*I − P*S*p_1))))
    THEN   TRUE   ELSE   epsilon(bool_pred) ENDIF
```

**Fig. 4.** Feedback Example.

end was reached on some branch of an induction proof. But, unlike the minor errors, making simple corrections to the specifications or to the statements of the lemmas would not remedy the problems. In these cases, close examination of the corresponding parts of the original hand proofs revealed errors in reasoning.

We note that the discovery of an error in the hand proof does not necessarily mean that a given assertion is false. It merely means that the assertion does not hold for the stated reasons. At least one invariant lemma in [12] (lemma 6) in fact had a *simpler* PVS proof than the hand proof supplied. When assertions hold for reasons other than the expected ones, one has to wonder whether the specification has in fact captured the required concepts.

Of the three modes of error discovery, the typechecking process produced the most varied results. Remedying the error in one case required extending the effect of an operator; in another, it required introducing a new invariant. A third "error" was one uncovered by the typechecker during a proof via the TCC (type-correctness condition) mechanism: when the $\epsilon$*−axiom* was invoked to assert the corresponding predicate on the nondeterministic water-level component $q'$ of the result state of the time passage action, a TCC was generated that required proving that $q'$ belongs to the correct subrange of values, namely [0, **C**]. "Error" is in quotes because this problem can be avoided by letting the water level range over all real values.[5] Provided that it can be established, Theorem 2 will then show that the water level does in fact stay within the physical limits.

**Time Required to Check the Boiler Controller.** Checking the Boiler Controller solution required the checking of 14 lemmas and 2 theorems. The entire process of encoding three specifications and checking the proofs (with backtracking when corrections were made in the specification or statements of the lemmas) took less than 3 work weeks. Part of this time was spent designing and implementing modifications to our system to support reasoning about nonlinear real arithmetic and nondeterminism. In a few proofs, we did not complete the check

[5] However, the Boiler automaton as specified is then not a priori constrained in the same way as the physical Boiler.

of the real arithmetic inequalities when this became too tedious in PVS and when their validity became obvious.

When all went smoothly, checking the proof of an invariant required on average half an hour to an hour. Exceptions occurred in three induction proofs involving significant reasoning about complex inequalities. For these, our basic induction strategy typically took between 10 and 15 minutes to reach the stage where the user could continue the proof by completing the base case and action cases, as opposed to times more on the order of 30 seconds for other induction proofs of invariants.[6] The reason for this inefficiency is a question for further study.

## 5 A Proof Example

Most of the invariant proofs in our case study are very similar to the corresponding hand proofs. However, the PVS proofs required more detail when they involved either nondeterminism or sufficiently complicated nonlinear arithmetic. An example of a state invariant for the Boiler Controller specification in Figure 2 that involves both reasoning about a nondeterministic action and nonlinear arithmetic is Lemma 11 in [12]. Figure 5 provides the statement and hand proof of Lemma 11 from [12]. Figure 6 shows the corresponding PVS proof using our specialized strategies.

This is one case in which no special help was required to allow PVS to reason about the real arithmetic: this aspect of the proof can be done by applying equalities, expanding products, and matching the resulting terms, all of which the primitive proof steps of PVS can do automatically.

On the other hand, PVS does need help at the point where the constraints on the nondeterministic value $v'$, the steam rate in the result state, are introduced. The step in the PVS proof which does this is the **USE_EPSILON** step. The effect of this step is to invoke the $\epsilon$*–axiom* for the (parameterized) predicate $P =$ **steam_rate_pred(v(s_1), t_1)** and the type $\tau = \mathit{nonnegreal}$, thus creating the two proof branches represented by the cases "1" and "2" that follow this step. In the first branch, the required constraints on $v' = \epsilon(P)$ have been introduced into the hypotheses of the current goal. In the second branch, it is required to prove that there does indeed exist a possible value for $v'$—that is, that there is indeed a value of type *nonnegreal* satisfying $P$. This is done by supplying the instance $v(s\_1)$. This branch is necessary because of the form of the $\epsilon$*–axiom* (see Section 3.1).

## 6 Conclusions

Applying our proof system to a solution of the Steam Boiler Controller problem led to several insights about the utility of our approach to specifying and verifying hybrid systems. We discuss these insights below.

**Mechanizing Specifications and Proofs.** The results of our study clearly demonstrate the utility of mechanized tools for detecting *and* correcting errors in both formal specifications and formal proofs. While verification systems that use totally automatic proof methods can provide some limited feedback, the generality of interactive theorem proving provides more specific, and therefore more useful, feedback. The feedback we obtained from the PVS typechecker and

[6] These times are on a SPARCstation 20 running SunOS 5.4.

Lemma 11: In all reachable states of the combined steam boiler system,

$$v + U_1*(read - now) < W \text{ or } stop = \textbf{true}$$

**Proof.** The basis is vacuously satisfied. We distinguish on the cases for the action $a$. For $a \in$ {sensor, activate} this lemma is trivially true. Otherwise we get:

A) $a$ = actuator ($v$, $stop$, and $now$ are unchanged):
We know $sr + U_1*I < W$ or $stopmode$ = **true** (Lemma 3.3), $do\_output$ = **true** from the precondition and if $do\_output$ then $now = read$ and $sr = v$ (Lemma 4). From this we can infer $v + U_1*(now + I - now) < W$ or $stopmode$ = **true**. Moreover, we get $stop' = e\_stop = stopmode$ and $read' = now + I$ from the effect and thus, we know $v + U_1*(read' - now) < W$ or $stop$ = **true**.

B) $a$ = time-passage ($read$ and $stop$ are unchanged):
We know from the precondition $stop$ = **false** and $v + U_1*(read - now) < W$ from the assumption. This is equivalent to $v + U_1*(read - now - \Delta t + \Delta t) < W$ and it follows $v + U_1*\Delta t + U_1*(read - now - \Delta t) < W$. Since we know from the effect $v' \le v + U_1*\Delta t$ and $now' = now + \Delta t$, finally, this is equivalent to $v' + U_1*(read - now') < W$. □

**Fig. 5.** Hand Proof of Lemma 11

```
Inv_11(s: states): bool  =  (v(s) + U_1*(read(s) – dur(now(s))) < W  OR  stop(s) = true);

lemma_11: LEMMA ( FORALL (s: states): reachable(s) => Inv_11(s) );

("" (AUTO_PROOF_BOILERSYS "Inv_11")
 (("1" (APPLY (THEN (EXPAND "enabled_specific") (BOILERSYS_SIMP))
         "Case nu(t_1)." "Invoke the precondition.")
       (APPLY (THEN (USE_EPSILON "nonnegreal" "steam_rate_pred" "v(s_1)" "t_1") (BOILERSYS_SIMP))
         "Invoke the restrictions on v'.")
       (("1" (TIME_ETC_SIMP))
        ("2" (APPLY (THEN (EXPAND "steam_rate_pred" 1) (BOILERSYS_SIMP))
               "Doing the existence proof for epsilon.")
             (APPLY (THEN (INST 1 "v(s_1)") (BOILERSYS_SIMP)))
             (TIME_ETC_SIMP))))
  ("2" (APPLY (THEN (APPLY_INV_LEMMA "3_3") (BOILERSYS_SIMP))
         "Case actuator(e_1,p_1).")
       (APPLY (THEN (EXPAND "enabled_specific") (BOILERSYS_SIMP))
         "Invoke the precondition.")
       (APPLY (THEN (APPLY_INV_LEMMA "4") (BOILERSYS_SIMP))))))
```

**Fig. 6.** PVS Proof of Lemma 11

from reaching dead ends in proofs not only localized the source of an error but often made it easy for us to to correct the error. Moreover, the time required to check the specifications and the proofs was not significant.

**Using Specialized Tools.** We have applied the system that we developed for our initial study [2] to two additional problems, the Boiler Controller benchmark and the proof of a timed version of Fischer's algorithm [13]. In each of the later applications, our earlier-developed system greatly simplified the specification and proof process. Each new application of the system led to some additions: the automata in Fischer's algorithm were a special class of timed automata that represent MMT automata [17]; those in the current study involved nondeterminism and nonlinear real arithmetic. However, the needed enhancements required the addition of only a very small number of new proof strategies, sometimes with

specification conventions to support them. The total time required for solving each of the two later problems, which included the time needed to add the required enhancements, was less than three work weeks.

It is not clear when we will reach the point where applying our proof system requires no additions. Our experience so far is that one can build upon the previously developed system. Further, while we have not yet reached the point where our system can be applied by users who lack knowledge of PVS, we believe that use of our system reduces the time and effort that are required to check proofs, while providing meaningful feedback about errors.

**On Reasoning About Nondeterministic Automata.** For efficient reasoning about nondeterministic automata in PVS, our current solution is to represent the transition relation as a function by making (careful) use of the Hilbert $\epsilon$. We believe that incorporation of "ANY" and "SOME" constructs with accompanying skolemization and instantiation would be useful additions to PVS for reasoning about nondeterminism, since this would help solve the problem of the results of actions being represented in PVS as deterministic (though nonspecific).

The desire to represent the transition relation as a function also partly arises from the intuition that the intention in specifications such as the one in this study is that transitions should be enabled when their preconditions are satisfied: that is, that they can then "fire". Thus, the transition relation should be considered to be a many-valued (but never no-valued) function. Note, this interpretation involves an implicit proof obligation for any such specification: one must demonstrate the existence of at least one result state satisfying the constraints of every legal transition. We note that PVS naturally enforces this requirement, since it demands that all functions be total.

**On Using PVS.** While there are no complete decision procedures for reasoning about inequalities involving nonlinear real arithmetic, our experience so far indicates that the human guidance PVS is likely to need in reasoning about hybrid automata is the application of a small number of facts about the real numbers, together with control over the grouping of terms into factors. It should be possible to design specialized PVS strategies that will provide much of the needed guidance. These strategies fall into two categories: those that simplify application of lemmas about real arithmetic, and what could be called "naming" strategies, strategies that simplify the use of names for terms. The second type can also be helpful in simplifying the job of the human user of PVS, since they could also be used to manage expressions, such as **epsilon** expressions, that do not necessarily involve a sum of terms, but are simply long. The implementation of at least some of these strategies will require the use of certain enhancements to PVS that are in progress, such as the ability to name and track assertions. Whether strategies can be designed that execute with acceptable speed remains an open question.

## Acknowledgments

We wish to thank Gunter Leeb and Nancy Lynch for sharing a challenging example with us; Steve Garland, Gunter Leeb, Victor Luchangco, and Nancy Lynch for insightful discussions; and the anonymous reviewers for helpful comments.

## References


1. J.-R. Abrial, E. Boerger, and H. Langmaack. Preliminary report for the Dagstuhl-Seminar 9523: Methods for Semantics and Specification. Dagstuhl, June 1995.
2. M. Archer and C. Heitmeyer. Mechanical verification of timed automata: A case study. In *Proc. 1996 IEEE Real-Time Technology and Applications Symp. (RTAS'96)*. IEEE Computer Society Press, 1996.
3. M. Archer and C. Heitmeyer. Verifying hybrid systems modeled as timed automata: A case study. Technical report, NRL, Wash., DC, 1997. In preparation.
4. R. Boyer and J. Moore. *A Computational Logic*. Academic Press, 1979.
5. S. Campos, E. Clarke, and M. Minea. Analysis of real-time systems using symbolic techniques. In *Formal Methods for Real-Time Computing*, chapter 9. John Wiley & Sons, 1996.
6. M. J. C. Gordon and T. Melham, editors. *Introduction to HOL: A Theorem Proving Environment for Higher-Order Logic*. Cambridge University Press, 1993.
7. C. Heitmeyer and N. Lynch. The Generalized Railroad Crossing: A case study in formal verification of real-time systems. In *Proc., Real-Time Systems Symp.*, San Juan, Puerto Rico, Dec. 1994.
8. C. Heitmeyer and N. Lynch. The Generalized Railroad Crossing: A case study in formal verification of real-time systems. Technical Report MIT/LCS/TM-51, Lab. for Comp. Sci., MIT, Cambridge, MA, 1994. Also TR 7619, NRL, Wash., DC 1994.
9. C. Heitmeyer and D. Mandrioli, editors. *Formal Methods for Real-Time Computing*. Number 5 in Trends in Software. John Wiley & Sons, 1996.
10. T. Henzinger and P. Ho. Hytech: The Cornell Hybrid Technology Tool. Technical report, Cornell University, 1995.
11. R. P. Kurshan. *Computer-Aided Verification of Coordinating Processes: the Automata-Theoretic Approach*. Princeton University Press, 1994.
12. G. Leeb and N. Lynch. Proving safety properties of the Steam Boiler Controller: Formal methods for industrial applications: A case study. In J.-R. Abrial, et al., eds., *Formal Methods for Industrial Applications: Specifying and Programming the Steam Boiler Control*, vol. 1165 of *Lect. Notes in Comp. Sci.* Springer-Verlag, 1996.
13. V. Luchangco. Using simulation techniques to prove timing properties. Master's thesis, Massachusetts Institute of Technology, June 1995.
14. N. Lynch and F. Vaandrager. Forward and backward simulations – Part II: Timing-based systems. To appear in *Information and Computation*.
15. N. Lynch and F. Vaandrager. Forward and backward simulations for timing-based systems. In *Proc. of REX Workshop "Real-Time: Theory in Practice"*, volume 600 of *Lecture Notes in Computer Science*, pages 397–446. Springer-Verlag, 1991.
16. K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.
17. M. Merritt, F. Modugno, and M. R. Tuttle. Time constrained automata. In J. C. M. Baeten and J. F. Goote, eds., *CONCUR'91: 2nd Intern. Conference on Concurrency Theory*, vol. 527 of *Lect. Notes in Comp. Sci.* Springer-Verlag, 1991.
18. S. Owre, J. Rushby, N. Shankar, and F. von Henke. Formal verification for fault-tolerant architectures: Prolegomena to the design of PVS. *IEEE Transactions on Software Engineering*, 21(2):107–125, Feb. 1995.
19. N. Shankar, S. Owre, and J. Rushby. The PVS proof checker: A reference manual. Technical report, Computer Science Lab., SRI Intl., Menlo Park, CA, 1993.
20. J. Vitt and J. Hooman. Assertional Specification and Verification Using PVS of the Steam Boiler Control System. In J.-R. Abrial et al., editors, *Formal Methods for Industrial Applications: Specifying and Programming the Steam Boiler Control*, volume 1165 of *Lect. Notes in Comp. Sci.* Springer-Verlag, 1996.

# A Appendix. Constant and Variable Definitions

| Name | Type | Unit | Description |
|---|---|---|---|
| $I$ | *real, >0* | *s* | time between the periodical sensor reading |
| $S$ | *real, >0* | *s* | delay to activate pumps after the last sensor reading |
| $U_1$ | *real, >0* | $l/s^2$ | maximum gradient of the increase of the steam rate |
| $U_2$ | *real, >0* | $l/s^2$ | maximum gradient of the decrease of the steam rate |
| $M_1$ | *real, >0* | *l* | minimum amount of water before boiler becomes critical |
| $M_2$ | *real, >0* | *l* | maximum amount of water before boiler becomes critical |
| $W$ | *real, >0* | *l/s* | maximum steam rate before boiler becomes critical |
| $P$ | *real, >0* | *l/s* | exact rate at which one active pump supplies water to the boiler |
| **#pumps** | *int, >0* | | Number of pumps that can supply water to the boiler in parallel |
| $C$ | *real, >0* | *l* | Capacity of the boiler |

**Table 1.** Constants for the Boiler and Controller Models

| Name | Initial Value | Type | Values Range | Unit | Description |
|---|---|---|---|---|---|
| *now* | 0 | real | $[0, \infty)$ | *s* | current time |
| *pr* | 0 | integer | {0, ..., **#pumps**} | | number of pumps actively supplying water to the boiler |
| *q* | $\gg M_1$, $\ll M_2$ | real | $[0, C]$ | *l* | actual water level in the boiler |
| *v* | 0 | real | $[0, \infty)$ | *l/s* | steam rate of the steam currently leaving the boiler |
| *pr_new* | 0 | integer | {0, ... , **#pumps**} | | number of pumps that are supposed to supply water after the activation delay |
| *error* | 0 | integer | {0, ... , *pr_new* } | | number of pumps that fail to supply water to the boiler after activation |
| *do_sensor* | **true** | boolean | {**true**, **false**} | | enable a single sensor reading |
| *set* | $S$ | real | $[0, \infty)$ | *s* | next time the pumps change to the new settings |
| *read* | 0 | real | $[0, \infty)$ | *s* | next time the sensors will be read |
| *stop* | **false** | boolean | {**true**, **false**} | | flag whether emergency shut down is activated |
| *do_output* | **false** | boolean | {**true**, **false**} | | flag that enables the output. This represents a kind of program counter. |
| *stopmode* | **false** | boolean | {**true**, **false**} | | flag to activate the shut down |
| *wl* | *q* | real | $[0, C]$ | *l* | current water level reading |
| *sr* | 0 | real | $[0, W]$ | *l/s* | current steam rate reading |
| *now* | 0 | real | $[0, \infty)$ | *s* | current time |
| *pumps* | 0 | integer | {0, ... , **#pumps**} | | number of currently active pumps suppling water to the boiler |
| *px* | 0 | integer | {0, ... , **#pumps**} | | number of pumps that shall supply water next |

**Table 2.** Variables for the Boiler and Controller Models

| Name | Type | Unit | Value | Description |
|---|---|---|---|---|
| *max_pumps_after_set* | integer | | **#pumps** | maximum number of pumps that can supply water to the boiler after the delay considering the pump failure model. |
| *min_pumps_after_set* | integer | | 0 | minimum number of pumps that can supply water to the boiler after the delay considering the pump failure model. |
| *max_steam_water* $(sr)$ | real | *l* | $\max(0, (sr - U_2 * I/2) * I)$ | minimum amount of water that can evaporate into steam until the next sensor reading |
| *min_steam_water* $(sr)$ | real | *l* | $(sr + U_1 * I/2) * I$ | maximum amount of water that can evaporate into steam until the next sensor reading |

**Table 3.** Additional Definitions for the Controller Model

This article was processed using the LaTeX macro package with LLNCS style